1. **Iterative Deepening Search**

def perform\_depth\_limited\_search(tree, node, goal, depth\_limit):

if node == goal:

return True

if depth\_limit == 0:

return False

if depth\_limit > 0:

for child in tree.get(node, []):

if perform\_depth\_limited\_search(tree, child, goal, depth\_limit - 1):

return True

return False

def iterative\_deepening\_search(tree, start, goal):

depth\_limit = 0

while True:

if perform\_depth\_limited\_search(tree, start, goal, depth\_limit):

return True

else:

depth\_limit += 1

if \_\_name\_\_ == "\_\_main\_\_":

# Build the tree structure as a dictionary

tree = {

"S": ["A", "C"],

"A": ["D", "B"],

"C": ["E", "G"],

"D": ["F", "H"],

"E": ["I"],

"B": [],

"G": [],

"F": [],

"H": [],

"I": [],

}

# Replace 'goal\_node' with your actual goal node value.

goal\_node = "G"

if iterative\_deepening\_search(tree, "S", goal\_node):

print(goal\_node + " Goal found " )

else:

print("Goal not found.")

1. **Depth Limited Search**

def depth\_limited\_search(tree, node, goal, depth\_limit):

if node == goal:

return True

if depth\_limit == 0:

return False

if node not in tree:

return False

for child in tree[node]:

if depth\_limited\_search(tree, child, goal, depth\_limit - 1):

return True

return False

if \_\_name\_\_ == "\_\_main\_\_":

# Build the tree structure as a dictionary

tree = {

"X": ["A", "B"],

"A": ["C", "D"],

"B": ["I", "J"],

"c": ["E", "F"],

"D": ["G"],

"I": ["H"],

"J": [],

"E": [],

"F": [],

"G": [],

"H": [],

}

# Replace 'goal\_node' with your actual goal node value.

goal\_node = "H"

depth\_limit = 2 # Set the depth limit according to your requirements.

if depth\_limited\_search(tree, "X", goal\_node, depth\_limit):

print("Goal found at " + goal\_node)

else:

print("Goal not found.")

1. **Binary Tree**

def create\_node(current, left=None, right=None):

return [current, left, right]

def preorder\_traversal(node):

if node is not None:

print(node[0], end=' ')

preorder\_traversal(node[1])

preorder\_traversal(node[2])

def inorder\_traversal(node):

if node is not None:

inorder\_traversal(node[1])

print(node[0], end=' ')

inorder\_traversal(node[2])

def postorder\_traversal(node):

if node is not None:

postorder\_traversal(node[1])

postorder\_traversal(node[2])

print(node[0], end=' ')

if \_\_name\_\_ == "\_\_main\_\_":

root = create\_node('a',

left=create\_node('b',

left=create\_node('c'),

right=None),

right=create\_node('d',

left=create\_node('e'),

right=create\_node('f')),

)

print("Pre-order Traversal:", end=" ")

preorder\_traversal(root)

print("\nIn-order Traversal:", end=" ")

inorder\_traversal(root)

print("\nPost-order Traversal:", end=" ")

postorder\_traversal(root)